(12) UK Patent Application (19) GB (11) 2 342 732 (13) A

(43) Date of A Publication 19.04.2000

(21) Application No 9822472.8

(22) Date of Filing 16.10.1998

(71) Applicant(s)
International Business Machines Corporation
(Incorporated in USA - New York)
Armonk, New York 10504, United States of America

(72) Inventor(s)
Paul Mundy

(74) Agent and/or Address for Service
IBM United Kingdom Limited
Intellectual Property Department, Mail Point 110, Hursley Park, WINCHESTER, Hampshire, SO21 2JN, United Kingdom

(51) INT CL[7]
G06F 7/00

(52) UK CL (Edition R)
G4A AAU

(56) Documents Cited
US 4716541 A **US 4298933 A**

(58) Field of Search
UK CL (Edition Q) **G4A AAU ACX**
INT CL[6] **G06F**

(54) Abstract Title
Reevaluation of a Boolean function applicable to event driven transaction processing

(57) A method, apparatus and computer program product reevaluates a Boolean function such as may arise as triggering events in the data processing of long running business transactions. The Boolean function consists of multiple groups of component Boolean functions which may be represented as respective nodes in a hierarchy and whose inputs may be externally input Boolean values or the Boolean values of other groups lower in the hierarchy, the current values of each of the groups and of the overall function being known in advance. The method stores for each group its current value then, in response to a change of any external input, reevaluates the value of any group to which the external input is applied. If the group value has changed, the reevaluation is repeated for any present group receiving that group value as input until the value of the respective group does not change or until there is no parent group. In this way, the overall value of the Boolean function may be reevaluated without a total recalculation.

GB 2 342 732 A

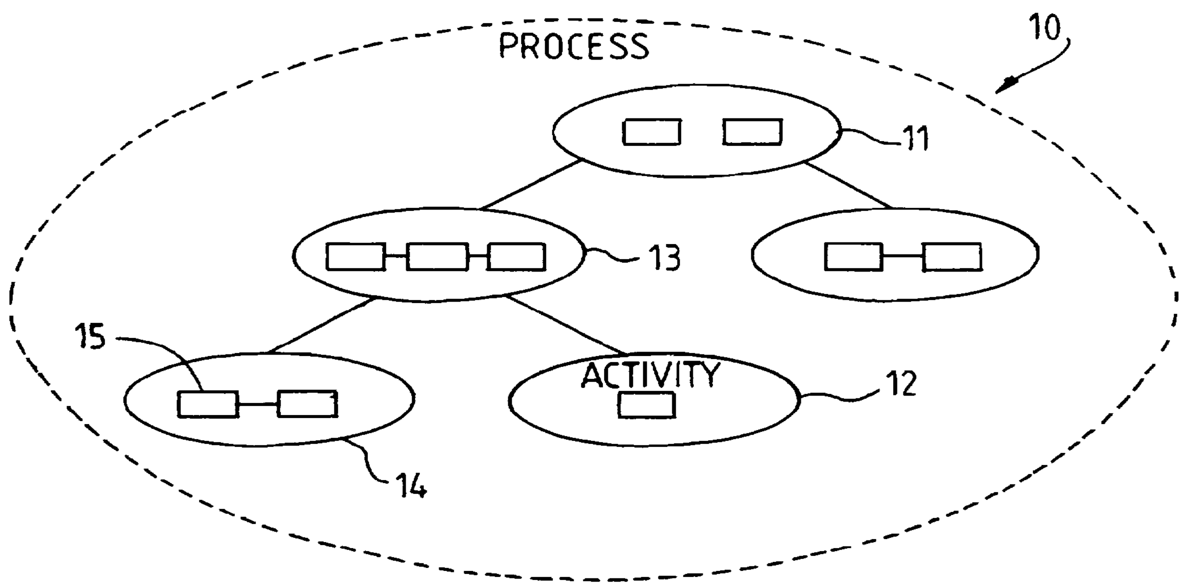

FIG. 1

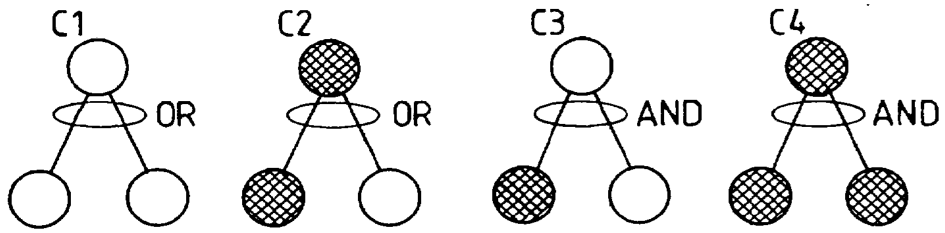

FIG. 2

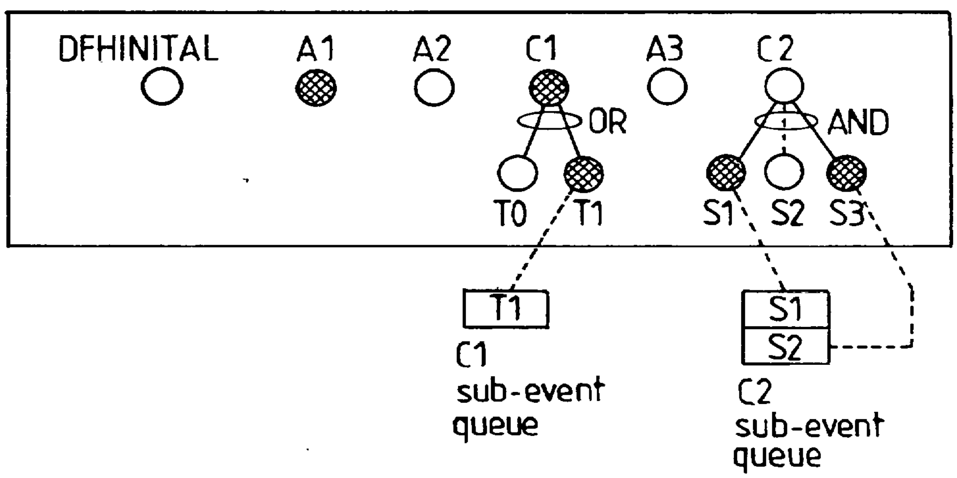

FIG. 3

EVENT NAME
LOGIC TYPE (AND or OR)
FIRE STATUS (FS)
PARENT ADDRESS (Parentp)
N1
N2

FIG. 4

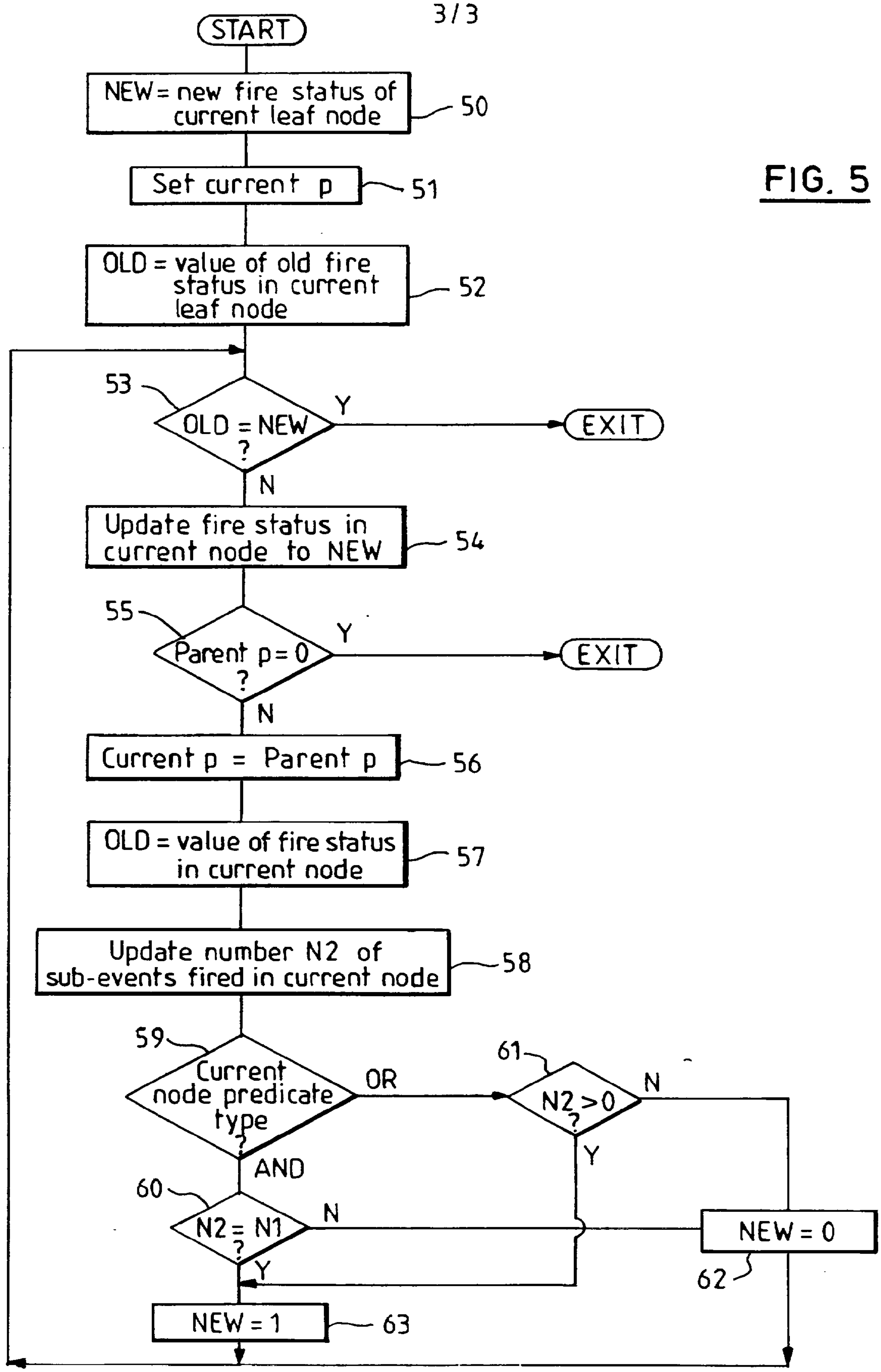
START
NEW = new fire status of current leaf node
50
Set current p
51
OLD = value of old fire status in current leaf node
52
53
OLD = NEW ?
Y
EXIT
N
Update fire status in current node to NEW
54
55
Parent p = 0 ?
Y
EXIT
N
Current p = Parent p
56
OLD = value of fire status in current node
57
Update number N2 of sub-events fired in current node
58
59
Current node predicate type ?
OR
61
N2 > 0 ?
N
Y
AND
60
N2 = N1 ?
N
NEW = 0
62
Y
NEW = 1
63

FIG. 5

2342732

METHOD AND APPARATUS FOR REEVALUATION OF A BOOLEAN FUNCTION APPLICABLE TO EVENT DRIVEN TRANSACTION PROCESSING

Technical field of the Invention

The present invention relates to methods and apparatus for the reevaluation of a Boolean function in response to a change of any input and also to a method and system of event driven transaction processing which includes reevaluation of an event state.

Background of the Invention

In many areas of data processing and computing, complex Boolean functions arise, the value of which needs to be known to trigger the next step of processing. The value of the expression (1 or 0) depends of course on the logical function but also on the values of its external inputs, such as the results of comparisons, yes/no inputs in response to questions to human users, the expiry of time intervals and many other possibilities. Complex Boolean functions also contain nested component functions which must be evaluated before the entire expression can be evaluated. There can be many layers of nesting. An alternative representation, useful in describing the present invention, is to regard the component expressions as being arranged in hierarchical fashion, with the external inputs at the bottom of a logical tree and the output value of the entire expression at the top.

Where such complex logic with many inputs is involved, the speed of processing may be reduced by the need to repeatedly evaluate the value of the Boolean function and the time each such reevaluation takes.

Techniques to speed up data processing may be employed to minimise the time taken for this calculation. For example, in UK patent application GB 2284690A, a computer system and method for evaluating predicates and Boolean expressions is described in which the values of component Boolean predicates are preset and changed in a predefined manner in accordance with the result of each operation. The system also uses parallel processing techniques. However, the entire expression still needs to be reevaluated every time there is a change.

US Patent 5291612 describes a technique for evaluating Boolean expressions using a tree of partial subfunction differentials to produce a total differential for the entire function. Redundant processing can

be avoided if the values of intermediate total differentials which constitute the input to the total differential for the entire function, all become zero. This method of evaluation is contrasted with methods which calculate actual function and subfunction.

One area in which evaluation of Boolean expressions can become critical is in the area of event driven transaction processing, which, of course is a real time process. In attempting to extend traditional transaction processing to long running business transactions, it becomes necessary to define a very large number of events which can trigger transaction processing activities and the triggering event can be a complex Boolean function of a number of subevents, which can either be single external inputs or can themselves be composite events which are Boolean functions of yet lower order events. As the processing of the long running transaction proceeds, and subevents occur, these expressions must be constantly and rapidly reevaluated to determine whether the event is to be "fired" to trigger some processing action.

There is therefore a need to minimise the time it takes to reevaluate such expressions.

Disclosure of the Invention

Accordingly, the present invention provides a method of reevaluating a Boolean function consisting of multiple groups of component Boolean functions which may be represented as respective nodes in a hierarchy and whose inputs may be externally input Boolean values or the Boolean value of another group lower in the hierarchy, the current values of each of the groups and of the overall function being known in advance; said method comprising storing for each group its current value; in response to a change of any external input, reevaluating the value of any group to which said external input is applied; and if the group value has changed, repeating the reevaluation for any parent group receiving said group value as input until the value of the respective group does not change or until there is no parent group, whereby the overall value of the Boolean function may be reevaluated without a total recalculation.

The invention also provides a method of data processing long running business transactions in which component activities of the transaction may be triggered by the firing of corresponding top events, which events may themselves be fired by a logical combination of events, including individual (leaf) events and composite events which may be

represented as respective nodes in a tree structure; the method comprising the steps of initially setting the states of a top event and its events to predefined Boolean values; storing said predefined values; detecting individual event changes; in response to each such change, reevaluating the value of any composite event to which said individual event contributes; and if the composite event value has changed, repeating the reevaluation for any parent event in said tree until the value of the parent event does not change or until there is no further parent event, whereby the state of a top event may be reevaluated without a total recalculation of all its components.

Thus the invention affords a faster reevaluation of the value of a logical function which in turn enables faster processing of transactions which are dependent on multiple subevents in a predefined logical combination to trigger processing stages. The increase in speed is achieved because only those portions of the logical tree affected by the changed input need be reevaluated. Reevaluation involves only straightforward calculation of group values and is self-limiting without the need to calculate partial and total differentials.

Preferably, the component Boolean functions are AND and OR functions, said storing step further comprises storing the total number, N1, of direct inputs and the number, N2, of such inputs taking a first Boolean value,and the reevaluation step comprises determining whether N1 equals N2 if the group is an AND group, corresponding to the AND function being satisfied, and determining whether N2 is greater than zero if the group is an OR group, corresponding to the OR function being satisfied.

This is a simple and fast way of evaluating the logical value of each function from values stored in a data structure.

In the transaction processing method, according to the invention, the initial setting of the events to predefined values is preferably to zero. After reevaluation, the current event values are stored instead.

In the preferred arrangement of determining the values of AND and OR functions,not only the current event values but also the numbers N1 and N2, the type of Boolean functions and a pointer to any parent event or group is stored in the data structure. If there is no parent, as at the top of the tree, the pointer value is set to zero.

Reevaluation may be triggered not only by changes in the value of an existing external input but also by the system adding or deleting further inputs from the logical tree. In this case, the data structures for the composite events must be updated.

If each external input is unique to one overall Boolean function or event, then the logic tree to be reevaluated is immediately apparent. If, on the other hand, the input is supplied to more than one function, some sort of mapping table will be needed to determine which trees need reevaluation.

The invention also provides both data processing apparatus and a computer program product for carrying out the steps of the above method.

Brief Description of the Drawings

The invention will now be described, by way of example only, with reference to a preferred embodiment thereof, as illustrated in the accompanying drawings, in which :

Figure 1 is an overview of a system for processing Activities which are part of long running business transactions;

Figure 2 illustrates basic types of composite Boolean events which may trigger Activities in the system of Figure 1;

Figure 3 shows an Event Pool for an Activity, including both composite and individual events;

Figure 4 shows a control block for storing information about a composite event; and

Figure 5 is a flow diagram of a method for reevaluating the Boolean value of a composite event according to the invention.

Detailed Description of the Invention

The environment in which the preferred embodiment of the invention is implemented is in a system for processing long running business transactions. A business transaction is a self-contained business deal of which purchasing a vacation is an example. This may involve multiple actions by a travel agent over an extended period involving, for example, flight bookings, hotel reservations, car reservations, invoicing and

payment processing. Although the transaction is viewed by agent and customer as a single whole, it is in fact a series of individual actions which take varying lengths of time, may be optional or may be susceptible to failure.

Traditional transaction processing is not well suited to dealing with such long running transactions because the "transactions" with which it operates are short discrete data processing actions consisting of one or more units of work which, in order to ensure recoverability in the case of failure, have to maintain locks on resources while they are in process. Holding locks for extended periods would not be feasible because system performance would suffer or the system would deadlock.

However, traditional transactions can be used as building blocks in the execution of a long-running business transaction by a Business Transaction Service (BTS), the structure of which is shown in Figure 1.

ACTIVITIES

In the BTS, an initial request starts a Process 10 corresponding to the entire business transaction. The process 10 consists of a collection of Activities which are the basic units of BTS execution. Activities can be hierarchically organized, in a tree structure. An activity 13 that starts another activity is known as a "parent activity". An activity 14 that is started by another is known as a " child activity". A root activity 12 is always at the top of an activity tree. A process always contains a root activity. When a process is started, the program that implements its root activity receives control. Typically, a root activity is a parent activity that creates and controls a set of child activities - that is, it manages their ordering, concurrent execution, and conditional execution. A root activity also controls synchronization, parameter passing and saving of state data. Within each activity may be traditional short lived transactions 15, such as are processed by the CICS family of transaction servers from IBM (CICS is a registered trademark of International Business Machines Corporation).

To complete its entire work, an activity may need to execute as a sequence of separate processing steps, or activations. For example, a parent activity typically needs to execute for a while, finish execution temporarily, then continue execution when one of its children has completed.

Each activation is "triggered" by a BTS event, and consists of a single transaction 15. A BTS event is a means by which BTS signals progress in a process. It informs an activity that an action is required or has completed. "Event" is used in its ordinary sense of "something that happens". An activity's first activation is triggered by a system event supplied by BTS after the first RUN or LINK command is issued against the activity. When the last activation ends, the activity completion event is "fired", which may, in turn, trigger another activity's activation.

A named area of storage, associated with a particular process or activity, and maintained by BTS is known as a data container. Each process or activity can have any number of data-containers. They are used to hold state data, and inputs and outputs for the activity.

A timer (not shown in Figure 1), is a BTS object that expires when the system time becomes greater than a specified date and time, or after a specified period has elapsed. Each timer has an event associated with it. The event occurs ("fires") when the timer expires. A timer can be used, for example, to cause an activity to be invoked at a particular time in the future.

Thus, within the BTS, activities, data containers and timers are used to manage many business transactions (processes), record the current status of each business transaction and to ensure that each activity is invoked at the appropriate times.

An activity is always in one of the following processing states or modes:

"ACTIVE"

- An activation of the activity is running.

"CANCELLING"

- The activity is waiting to be cancelled (a CANCEL ACTIVITY command has been issued).

"COMPLETE"

- The activity has completed, either successfully or unsuccessfully.

"DORMANT"

- The activity is waiting for an event to fire its next activation - which could be its first, or a subsequent, activation.

"INITIAL"

- No RUN or LINK command has yet been issued against the activity; or the activity has been reset to its initial state by means of a RESET ACTIVITY command.

EVENTS

As stated above, a BTS event is a means by which CICS business transaction services signal progress in a process. It informs an activity than an action is required or has completed. To define an event it is given a name. An activity program uses such commands as DEFINE INPUT EVENT, DEFINE TIMER, and the EVENT option of DEFINE ACTIVITY to name events about which it wants to be informed.

Named events have Boolean values - FIRED or NOTFIRED. When first defined, an event has the NOTFIRED value. When an event occurs it is said to *fire* (that is to make the transition from NOTFIRED to FIRED). An activity can, for example:

- Discover the event (or events) whose firing caused it to be reattached (RETRIEVE REATTACH EVENT)

- Test whether an event has fired (TEST EVENT).

BTS events can be *atomic* or *composite*.

Atomic events

An atomic event is a single, "low-level" occurrence (which may happen under the control of BTS or outside the control of BTS). There are four types of atomic event:

- Input events
- Activity completion events
- Timer events
- System events

Input events

Input events tell activities why they are being run. A RUN or LINK ACTIVITY command delivers an input event to an activity, and thus activates the activity. (The INPUTEVENT option on the command *names* the input event and thus defines it to the requestor).

The first time an activity is run, a system event, DFHINITIAL, tells the activity to perform its initial housekeeping. Typically, this involves defining further events for which it may be activated.

An activity must use the RETRIEVE REATTACH EVENT command to discover the event or events that caused it to be activated. On any activation (but typically on its first, when it is invoked with DFHINITIAL), it may use the DEFINE INPUT EVENT command to define some input events for which it can be activated subsequently.

Activity completion events

The completion of a child activity (but not a root activity) causes the activity completion event to fire. (The EVENT option on the DEFINE ACTIVITY command names the activity completion event and thus defines it. If EVENT is not specified, the completion event is given the same name as the activity itself).

Timer Events

As explained above, a timer is a BTS object that expires when the system time becomes greater than a specified date and time, or after a specified period has elapsed. When you define a timer, a timer event is automatically associated with it. When the timer expires, its associated event fires.

To define a timer, a command is issued. A timer that specifies a date and time that has already passed expires immediately. Similarly, if the requested interval is zero, the timer expires immediately.

To check whether a timer has expired and, if it has, whether it expired normally a CHECK TIMER command is used.

System events

All the other types of event described in this section (including composite events) are referred to as user-defined events, because they are defined by the BTS application programmer, using commands such as

DEFINE INPUT EVENT, and DEFINE TIMER. BTS system events, on the other hand, are defined by BTS. They are a special kind of input event. There is only one type of BTS system event - DFHINITIAL, discussed above.

Composite events

A composite event is a "high-level" event, formed from zero or more user-defined (that is non-system) atomic events. When included in a composite event, an atomic event is known as a subevent.

A DEFINE COMPOSITE EVENT command defines a *predicate*, which is a logical expression typically involving subevents. At all times, the composite event's fire status reflects the value of the predicate. When the predicate becomes true, the composite event fires; when it becomes false, the composite's fire status reverts to NOTFIRED.

The logical operator that is applied to the composite event's predicate is one of the Boolean operators AND or OR. *AND and OR cannot both be used.*

When first defined, a composite event is "empty" - that is, it contains no subevents. An ADD SUBEVENT command can be used to add subevents to the empty composite event. A composite event that uses the OR Boolean operator fires when *any* of its subevents fires. A composite event that uses the AND operator fires when *all* of its subevents have fired.

Figure 2 shows four composite events, C1 through C4. Each composite event contains two subevents. C1 and C2 use the OR Boolean operator. C3 and C4 use the AND operator. The shaded circles indicate the events that have fired.

The convention is adopted that an empty composite event that uses the AND operator is always true (FIRED). An empty composite event that uses the OR operator is always false (NOTFIRED).

The subevent queue

The names of subevents that fire are placed on the composite event's subevent queue - from where they can be retrieved by issuing one or more RETRIEVE SUBEVENT commands. Each composite event has a subevent queue associated with it. The subevent queue may be empty or may contain only the names of those subevents that have fired and not been retrieved

Event Pool

Figure 3 shows an Event Pool containing all the events that are recognized by a particular activity. Among them are two composite events, C1 and C2. The subevent queue for C1 contains the name T1. The subevent queue for C2 contains the names S1 and S3.

Each activity has an event pool, which contains the set of events that it recognizes. The events that an activity recognizes are:

1. Events that have been defined to it by means of:
   - DEFINE COMPOSITE EVENT
   - DEFINE INPUT EVENT
   - DEFINE TIMER
   - The EVENT option of the DEFINE ACTIVITY command.

2. System events.

An activity's event pool is initialized when the activity is created, and deleted when the activity is deleted. All event-related commands described operate on the event pool associated with the *current* activity.

Events can be deleted (that is, both the event and its name discarded). If the event is a subevent, the value of the composite event will be that of its predicate, after the subevent has been removed from the predicate's Boolean expression.

The command used to delete an event depends on the type of event to be deleted:

- To delete an input event, use the DELETE EVENT command.
- To delete a composite event, use the DELETE EVENT command. Note that deleting a composite event does not delete the composite's subevents.
- An activity completion event is implicitly deleted when a response from the completed activity has been acknowledged by a CHECK ACTIVITY command issued by the activity's parent; or when a DELETE ACTIVITY command is issued.
- A timer event is implicitly deleted if its associated timer has expired and a CHECK TIMER command is issued by the activity that owns it; or when a DELETE TIMER command is issued.
- System events cannot be deleted.

Reattachment events and activity activation

An activity is reattached (reactivated) on the firing of any event (other than a subevent) that is in its event pool. In other words, an activity is reattached when either of the following types of event occurs:

- A user event that has been defined to the activity and not included in a composite event. The user-event may be:
  - An input event
  - The completion event for a child activity
  - A timer event
  - A composite event

- A system event.

An event that causes an activity to be reactivated is known as a reattachment event.

Note: The firing of a subevent never directly causes an activity to be reattached - it is the firing of the associated composite event that does so. Therefore, a subevent can never be a reattachment event.

Handling reattachment events

When an activity is reattached, it must use the RETRIEVE REATTACH EVENT command to discover the event that caused reattachment. At times reattachment may occur because of the firing of more than one event. The names of the reattachment events are placed on a queue - the reattachment queue - from where they can be retrieved by issuing one or more RETRIEVE REATTACH EVENT commands. Each activity has a reattachment queue, which:

- May be empty
- Contains only the names of those reattachment events that have fired and not been retrieved.

If the event that caused it to be reattached is composite, an activity should issue one or more RETRIEVE SUBEVENT commands to discover the subevent or subevents that fired.

Each time it is activated, an activity must deal with at least one reattachment event - that is, it must issue at least one RETRIEVE

REATTACH EVENT command. Failure to do so results in the activity completing abnormally. If there is more than one event on the reattachment queue and the activity returns without retrieving them all, it is immediately reactivated to deal with the remaining events.

Resetting and deleting reattachment events: Retrieving an atomic event from the reattachment queue causes the event's fire status to be reset to NOTFIRED. Retrieving a composite event from the reattachment queue does *not* reset the event's fire status to NOTFIRED, because a composite event is only reset when the predicate becomes false. Thus, if an activity program retrieves a composite event and instantly returns, the result will be immediate reattachment because nothing has altered. To prevent immediate reattachment, the activity program should issue one or more RETRIEVE SUBEVENT commands, to retrieve (and reset) the subevent or subevents that have fired.

To delete input and composite events from its event pool, the activity must issue DELETE EVENT commands.

Activity completion

An activity completes when it returns with no user events in its event pool.

When an activity issues an EXEC CICS RETURN command (without the ENDACTIVITY option):

If there are events on the reattachment queue
The activity is immediately reactivated to deal with the fired events.

If there are no events on the reattachment queue, then:
If there are unfired user events (or fired subevents) in the event pool,
the activity becomes dormant until a reattachment event occurs, or
If there are no user events in the event pool,
the activity completes normally.

Thus, before trying to complete, an activity should ensure that it has deleted all the user events in its event pool.

Using the ENDACTIVITY option of the RETURN command: Optionally, an activity program can use the ENDACTIVITY option of the EXEC CICS RETURN command to signal that it has completed all its processing steps and is not to be reactivated.

When an activity issues an EXEC CICS RETURN ENDACTIVITY command, then:

If there are no user events in the activity's event pool,
the activity completes normally, or
If there are user events (fired or unfired) in the activity's event pool,
the events are deleted and the activity completes "unexpectedly". A CHECK ACTIVITY command issued against the activity returns a completion status of UNEXPECTED.

Evaluation of Events

Although not illustrated in Figures 2 and 3, it is possible that the subevents of a composite event may themselves be composite events and so on. In this case a tree of Boolean expressions is created in which the lower level expressions provide inputs to the higher level expressions until the overall value emerges at the top of the tree. Each Boolean expression defining a composite event is a predicate, the value of which determines the fire status of the composite event.

For example, where a composite event defines a predicate which is composed of atomic events, X, Y and Z, a composite event C may be defined as: C = (X & Y) | Z, where "&" is a logical AND and "|" is a logical OR.

Thus, the composite event C is fired if both X and Y are fired, or Z is fired, otherwise it is not-fired.

In such systems, predicate expressions may be complex and contain a large number of atomic events, the fire-status of any of the atomic events may change at any time, or the predicate expression itself may be changed (when at atomic event is added or deleted, for example).

The time taken to evaluate a complete expression of this type can be significant. More importantly, in an event driven system, with large numbers of events constantly changing, there is a frequent need for

reevaluation of complex events to determine if they have fired and whether a dependent activity should be started or reattached.

It is therefore important both to minimise the pathlength required to determine the fire-status of the composite at any time, and to minimise the pathlength required to update the fire-status of the composite when the fire status of an atomic event changes or the predicate expression itself changes.

The solution according to one embodiment of the present invention is as follows. To simplify the solution description initially, assume that a predicate may only consist of a simple 'AND' or 'OR' of a set of atomic events, for example:

C1 = (X & Y & Z)

C2 = (P | Q)

The predicate expression for a composite event can be represented as a tree structure, with each node in the tree containing a pointer to its parent or zero if it is the root node).

```
C1   =     &                    C2   =     |
          / | \                           / \
         X  Y  Z                         P   Q
```

Each event (atomic or composite) is managed by a control block containing:

- the event name
- the event's fire status
- the address of the event's parent (or zero, if is a root).

In addition, a composite event control block, as shown in Figure 4, includes the type of logical operator, the number of subevents, N1, and the number of subevents fired, N2.

The fire-status of a composite event can thus be determined as follows:

- If the predicate is 'AND', its fire-status is fired if the number N2 of subevents fired is equal to the number N1 of subevents, otherwise it is not-fired.
- If the predicate is 'OR', its fire-status is fired if the number N2 of subevents fired is greater than zero, otherwise it is not-fired.

When an atomic event is set fired or not-fired, its fire status is checked to see whether the fire-status of the event is being changed. If

not, then there is nothing more to do. If it is being changed, however, the 'number of subevents fired' N2 must be updated in the parent as necessary. The appropriate test to set the parent's fire status can then be made, depending whether it is an 'AND' or 'OR'.

The design can be extended to remove the restriction of having just a simple 'AND' or 'OR' to allow, for example:

```
C    =     |
          / \
         &    Z
        / \
       X   Y
```

With this extension to the design, a composite event can have another composite event as a subevent. Now if the fire-status of an atomic event changes, its containing composite's fire-status must be re-evaluated. If it has not changed, there is nothing more to be done. If it has change, however, its parent's fire-status must be reevaluated, and so on. The process stops when either there is no change at the current level, or the root is reached.

It can be seen that handling the addition or deletion of an atomic event requires similar processing, except that the number of subevents N1 in the containing composite must also be updated.

An implementation of the invention for reevaluating a predicate when the value of a leaf node is changed, is shown in Figure 5.

As illustrated in Figure 4, each node has a control block containing:

The fire status (0 or 1)
Parentp (address of parent or zero if this the root)
Predicate type (AND/OR) for a non-leaf node,
Number of subevents (for a non-leaf node), N1
Number of subevents fired (for a non-leaf node), N2.

Work variables used in predicate reevaluation are:

OLD	- old fire status
NEW	- new fire status
Currentp	- address of current node

Initially, these are set as follows:

Currentp - address of leaf node being updated

NEW - fire status to which leaf node is to be set.

Reevaluation of a Boolean expression is triggered when the fire status of one of the leaf node subevents contributing to the expression changes. At that time, the variable NEW is initialised to the new fire status of the leaf node in step 50 and the variable Currentp is set, in step 51, to the address of the composite event C to which the leaf node is an input.

In step 52, the OLD variable is set to the fire status of the current leaf node, prior to the reevaluation. In Step 53, if OLD = NEW no change is necessary and the reevaluation process ceases.

In step 54, the fire status in the current node (the leaf node) is updated to the value of NEW. In Step 55, if the address of the parent Parentp is zero, no further evaluation is needed as the root has been reached.

In step 56, Currentp is set to Parentp, that is, the node address of the composite event. In step 57, OLD is reset to the value of the fire status in the new current node (the composite event). Then the number N2 of subevents fired in the composite event node C is updated in step 58.

The next stage of the process is the determination at step 59 of whether the logic type of the composite event is AND or OR. If it is AND, then, at step 60, it is determined whether N2 = N1 and the AND is satisfied. If is an OR, then, at step 61, it is determined whether N2 is greater than zero and the OR is satisfied. NEW is then set to 0 or 1 as appropriate at steps 62 and 63.

The program then returns to step 53 and if OLD = NEW, the process terminates. If, however, there is a difference the reevaluation continues and the fire status is updated to NEW in step 54.

Thus all changes consequent upon the original leaf node change are worked through the process but portions of the expression which are unaffected are not reevaluated. This increases the efficiency of

reevaluation of Boolean expressions in general and the performance of event driven systems in particular.

CLAIMS

1. A method of reevaluating a Boolean function consisting of multiple groups of component Boolean functions which may be represented as respective nodes in a hierarchy and whose inputs may be externally input Boolean values or the Boolean value of another group lower in the hierarchy, the current values of each of the groups and of the overall function being known in advance;

said method comprising

storing for each group its current value;

in response to a change of any external input, reevaluating the value of any group to which said external input is applied; and

if the group value has changed, repeating the reevaluation for any parent group receiving said group value as input until the value of the respective group does not change or until there is no parent group, whereby the overall value of the Boolean function may be reevaluated without a total recalculation.

2. A method as claimed in claim 1 in which the component Boolean functions are AND and OR functions; in which said storing step further comprises storing the total number, N1, of direct inputs and the number, N2, of such inputs taking a first Boolean value; and in which said reevaluation step comprises determining whether N1 equals N2 if the group is an AND group, corresponding to the AND function being satisfied, and determining whether N2 is greater than zero if the group is an OR group, corresponding to the OR function being satisfied.

3. A method as claimed in claim 2 in which the current value and the numbers N1 and N2 for each group are stored in a data structure which also contains the type of Boolean function for said group and a pointer to the parent group, said method including the further step of identifying any parent group for reevaluation from said pointer.

4. A method as claimed in claim 3 in which the pointer is set to zero if there is no parent group, said method including the further step of

determining from the zero value of said pointer that there is no parent group for reevaluation.

5. A method of data processing long running business transactions in which component activities of the transaction may be triggered by the firing of corresponding top events, which events may themselves be fired by a logical combination of subevents, including individual (leaf) events and composite events which may be represented as respective nodes in a tree structure;

the method comprising the steps of

initially setting the states of a top event and its subordinate events to predefined Boolean values;

storing said predefined values;

detecting individual event changes;

in response to each such change, reevaluating the value of any composite event to which said individual event contributes; and

if the composite event value has changed, repeating the reevaluation for any parent event in said tree until the value of the parent event does not change or until there is no further parent event, whereby the state of a top event may be reevaluated without a total recalculation of all its components.

6. A method as claimed in claim 5 in which the initial setting step comprises setting the states of all events to zero.

7. A method as claimed in either claim 5 or claim 6 including the step of storing current event values after reevaluation.

8. A method as claimed in claim 7 in which the composite events are either AND or OR functions and which includes the further step of storing the total number, N1, of direct input events to the composite event and the number, N2, of such input events taking a first Boolean value; said reevaluation step comprising determining whether N1 equals N2 if the composite event is an AND function, corresponding to the AND function

being satisfied, and determining whether N2 is greater than zero if the group is an OR group, corresponding to the OR group being satisfied.

9. A method as claimed in claim 8 in which the current value and the numbers N1 and N2 for each composite event are stored in a data structure which also contains the type of Boolean function for said event and a pointer to any parent event, said method including the further step of identifying said parent event for reevaluation from said pointer.

10. A method as claimed in claim 9 in which the pointer is set to zero if there is no parent group, said method including the further step of determining from the zero value of said pointer that there is no parent group for reevaluation.

11. A method as claimed in any one of claims 5 to 10 in which the step of detecting individual event changes includes detecting addition or deletion of events from the logical tree, as well as changes in event value.

12. Apparatus for reevaluating a Boolean function consisting of multiple groups of component Boolean functions which may be represented as respective nodes in a hierarchy and whose inputs may be externally input Boolean values or the Boolean value of another group lower in the hierarchy, the current values of each of the groups and of the overall function being known in advance;

said apparatus comprising

storing means for storing for each group its current value;

reevaluation means responsive to a change of any external input to reevaluate the value of any group to which said external input is applied; and

repetition means responsive to a change in the group value, to cause said reevaluation means to repeat the reevaluation for any parent group receiving said group value as input until the value of the respective group does not change or until there is no parent group, whereby the overall value of the Boolean function may be reevaluated without a total recalculation.

13. Apparatus as claimed in claim 12 in which the component Boolean functions are AND and OR functions; in which said storing means is arranged to store the total number, N1, of direct inputs and the number, N2, of such inputs taking a first Boolean value; and in which said reevaluation means is arranged to determine whether N1 equals N2 if the group is an AND group, corresponding to the AND function being satisfied, and to determine whether N2 is greater than zero if the group is an OR group, corresponding to the OR function being satisfied.

14. Apparatus as claimed in claim 13 including a data structure in which the current value and the numbers N1 and N2 for each group are stored, said data structure also containing the type of Boolean function for said group and a pointer to the parent group, said apparatus further including means for identifying any parent group for reevaluation from said pointer.

15. A method as claimed in claim 14 in which the pointer is set to zero if there is no parent group, said apparatus further including means for determining from the zero value of said pointer that there is no parent group for reevaluation.

16. Apparatus for data processing long running business transactions in which component activities of the transaction may be triggered by the firing of corresponding top events, which events may themselves be fired by a logical combination of subevents, including individual (leaf) events and composite events which may be represented as respective nodes in a tree structure;

said apparatus comprising

initialising means for initially setting the states of a top event and its subordinate events to predefined Boolean values;

storing means for storing said predefined values;

detecting means for detecting individual event changes;

reevaluation means responsible to each such change, to reevaluate the value of any composite event to which said individual event contributes; and

repetition means responsible to a change in the composite event value, repeat the reevaluation for any parent event in said tree until the value of the parent event does not change or until there is no further parent event, whereby the state of a top event may be reevaluated without a total recalculation of all its components.

17. A method as claimed in claim 14 in which the initialising means is arranged to set the states of all events to zero.

18. A method as claimed in either claim 16 or claim 17 in which the storing means is arranged to store current event values after reevaluation.

19. Apparatus as claimed in claim 18 in which the composite events are either AND or OR functions and which said storing means is arranged to store the total number, N1, of direct input events to the composite event and the number, N2, of such input events taking a first Boolean value; said reevaluation means is arranged to determine whether N1 equals N2 if the composite event is an AND function, corresponding to the AND function being satisfied, and to determine whether N2 is greater than zero if the group is an OR group, corresponding to the OR group being satisfied.

20. Apparatus as claimed in claim 19 including a data structure in which the current value and the numbers N1 and N2 for each composite event are stored, also containing the type of Boolean function for said event and a pointer to any parent event, said apparatus further including identifying said parent event for reevaluation from said pointer.

21. Apparatus as claimed in claim 20 in which the pointer is set to zero if there is no parent group, said apparatus further including determining from the zero value of said pointer that there is no parent group for reevaluation.

22. Apparatus as claimed in any one of claims 16 to 21 in which the detecting means is arranged to detect addition or deletion of events from the logical tree, as well as changes in event value.

23. A computer program product recorded on a medium for carrying out a method of reevaluating a Boolean function consisting of multiple groups of component Boolean functions which may be represented as respective nodes in a hierarchy and whose inputs may be externally input Boolean

values or the Boolean value of another group lower in the hierarchy, the current values of each of the groups and of the overall function being known in advance;

said method comprising

storing for each group its current value;

in response to a change of any external input, reevaluating the value of any group to which said external input is applied; and

if the group value has changed, repeating the reevaluation for any parent group receiving said group value as input until the value of the respective group does not change or until there is no parent group, whereby the overall value of the Boolean function may be reevaluated without a total recalculation.

24. A computer program product as claimed in claim 23 in which the component Boolean functions are AND and OR functions; in which said storing step further comprises storing the total number, N1, of direct inputs and the number, N2, of such inputs taking a first Boolean value; and in which said reevaluation step comprises determining whether N1 equals N2 if the group is an AND group, corresponding to the AND function being satisfied, and determining whether N2 is greater than zero if the group is an OR group, corresponding to the OR function being satisfied.

25. A computer program product as claimed in claim 24 in which the current value and the numbers N1 and N2 for each group are stored in a data structure which also contains the type of Boolean function for said group and a pointer to the parent group, said method including the further step of identifying any parent group for reevaluation from said pointer.

26. A computer program product as claimed in claim 25 in which the pointer is set to zero if there is no parent group, said method including the further step of determining from the zero value of said pointer that there is no parent group for reevaluation.

27. A computer program product recorded on a medium for carrying out a method of data processing long running business transactions in which component activities of the transaction may be triggered by the firing of

corresponding top events, which events may themselves be fired by a logical combination of subevents, including individual (leaf) events and composite events which may be represented as respective nodes in a tree structure;

the method comprising the steps of

initially setting the states of a top event and its subordinate events to predefined Boolean values;

storing said predefined values;

detecting individual event changes;

in response to each such change, reevaluating the value of any composite event to which said individual event contributes; and

if the composite event value has changed, repeating the reevaluation for any parent event in said tree until the value of the parent event does not change or until there is no further parent event, whereby the state of a top event may be reevaluated without a total recalculation of all its components.

28. A computer program product as claimed in claim 27 in which the initial setting step comprises setting the states of all events to zero.

29. A computer program product as claimed in either claim 27 or claim 28 including the step of storing current event values after reevaluation.

30. A computer program product as claimed in claim 28 in which the composite events are either AND or OR functions and which includes the further step of storing the total number, N1, of direct input events to the composite event and the number, N2, of such input events taking a first Boolean value; said reevaluation step comprising determining whether N1 equals N2 if the composite event is an AND function, corresponding to the AND function being satisfied, and determining whether N2 is greater than zero if the group is an OR group, corresponding to the OR group being satisfied.

31. A computer program product as claimed in claim 30 in which the current value and the numbers N1 and N2 for each composite event are

stored in a data structure which also contains the type of Boolean function for said event and a pointer to any parent event, said method including the further step of identifying said parent event for reevaluation from said pointer.

32. A computer program product as claimed in claim 31 in which the pointer is set to zero if there is no parent group, said method including the further step of determining from the zero value of said pointer that there is no parent group for reevaluation.

33. A computer program product as claimed in any one of claims 27 to 32 in which the step of detecting individual event changes includes detecting addition or deletion of events from the logical tree, as well as changes in event value.

The Patent Office

2b

Application No:	GB 9822472.8	**Examiner:**	Mike Davis
Claims searched:	1-33	**Date of search:**	22 March 1999

Patents Act 1977
Search Report under Section 17

Databases searched:

UK Patent Office collections, including GB, EP, WO & US patent specifications, in:

UK Cl (Ed.Q): G4A (AAU, ACX)

Int Cl (Ed.6): G06F

Other:

Documents considered to be relevant:

Category	Identity of document and relevant passage		Relevant to claims
A	US 4716541	(QUATSE) eg abstract	-
A	US 4298933	(SHIMOKAWA) eg abstract	-

X	Document indicating lack of novelty or inventive step	A	Document indicating technological background and/or state of the art.
Y	Document indicating lack of inventive step if combined with one or more other documents of same category.	P	Document published on or after the declared priority date but before the filing date of this invention.
		E	Patent document published on or after, but with priority date earlier than, the filing date of this application.
&	Member of the same patent family		